METHOD AND SYSTEM FOR THE OPTIMAL FORMATTING, REDUCTION AND COMPRESSION OF DEX/UCS DATA

CROSS REFERENCE TO RELATED APPLICATION

This application claims priority from U.S. Provisional Patent Application Serial No. 60/203,682, filed May 12, 2000, and entitled "METHOD AND SYSTEM FOR THE OPTIMAL FORMATTING, REDUCTION AND COMPRESSION OF DEX/UCS DATA."

TECHNICAL FIELD

The present invention relates generally to data formatting, reduction and compression. More particularly, the present invention relates to a data formatting, reduction and compression method and system for use in wireless and/or wireline communication networks.

BACKGROUND OF THE INVENTION

Over the past decade, vending machine manufacturers have developed new and innovative vending equipment in response to market needs and vending operator demands. These innovations have been, for the most part, adopted by the beverage vending industry. This trend has been influenced by the accelerating rate of technological innovation in the electronic and electro-mechanical component industry. The availability of new technologies has given vending machine manufacturers the tools to address many of the requirements of vending operators. Advances in electronics are now enabling the use of computer controls and data acquisition systems directly inside the vending machine. Some of the latest vending machines now make it possible for vending machine operators to download sales, inventory, and machine health information on-site onto portable computers or to transmit the vending machine information to a central operations location.

SUMMARY OF THE INVENTION

In accordance with the teachings of the present invention, a system and method are provided to allow users to extend their corporate enterprise systems into the field using wireless data technologies. The system and method offer information solutions for a wide variety of e-commerce services. One aspect of the present invention is based on an application services platform or network operations center (NOC) upon which users host their wireless-enabled enterprise applications. The NOC manages the complexities of the wireless data realm while providing users with seamless access to their field data and enabling the integration of hand held wireless devices into the system. The present invention may be efficiently used in vertical industries such as cold drink vending, fast food restaurants (fountain drinks), ice merchandising, printing and imaging. Horizontal industries which may benefit from the teachings of the present invention include refrigeration, field service, and end-customer enablement using wireless data.

The present invention is particularly useful as a wireless data solution for vending machines that makes use of narrowband wireless networks and Internet-based e-commerce application services (using Java, XML, WAP, etc.) to enable vending operators to improve their sales and reduce their operational costs.

Accordingly, a method for efficiently and cost effectively communicating data between a network operations center and a remote device is provided. The method may involve transmitting a request for data to at least one remote device. Upon receipt of the request for

data by the remote device, a current state for the remote device is preferably established. After accessing a previous state for the remote device, a delta value is then preferably calculated between the current state and the previous state for the remote device. The delta data is then written to a device response and the device response is sent to the network operations center for database updating. In a further embodiment, the delta data is compressed before transmission to the network operations center.

The present invention also provides a method and system for communicating information between a network operations center and a remote device. This method of communication preferably begins by transmitting at least one request for information to the remote device. Upon receipt of the request, records are selected from a data block based upon the request. The selected records are then preferably restructured according to a template prior to transmitting the restructured records to the network operations center. In a further embodiment, the method may also compress a delta value calculated between a current set of restructured records and a previously stored set of restructured records.

In another embodiment, the present invention provides a method for communicating information between a network operations center and a remote device. In this "call-in" mode, the method preferably includes selecting records from a data block communicatively coupled to the device. The selected records are then preferably restructured according to a template and a delta is calculated between the restructured records and a stored

set of records. Once the delta has been calculated, the delta is preferably transmitted to the network operations center.

In yet another embodiment, the present invention provides a system for acquiring data at a remote device and communicating between a network operations center and the remote device. In this preferred "call-in" system, the remote device is preferably operable to establish communications with the network operations center. The remote device is preferably further operable to select at least one record from a data block communicatively coupled to the device. Upon selection of the record, the remote device is preferably operable to restructure the record according to a template available to the remote device. Once the record has been restructured, the remote device preferably calculates a delta between the delta and a stored set of records. The remote device then preferably transmits the delta to the network operations center via a network.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete and thorough understanding of the present embodiments and advantages thereof may be acquired by referring to the following description taken in conjunction with the accompanying drawings, in which like reference numbers indicate like features, and wherein:

FIGURE 1 is a block diagram of a system for communicating between a remote device and a network operations center incorporating teachings of the present invention;

FIGURE 2 is a block diagram of one embodiment of a remote data acquisition system for vending machines according to the present invention;

FIGURES 3A - 3B illustrates a template form for restructuring a DEX file according to one embodiment of the present invention;

FIGURES 4-8 illustrate various scenarios of data transmission and processing according to one embodiment of the present invention;

FIGURES 9A - 9B is a flow chart illustrating one example of preferred processing performed by a remote device according to one embodiment of the present invention; and

FIGURES 10A - 10B is a flow chart illustrating one example of preferred processing performed by a network operations center according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

Preferred embodiments of the invention and its advantages are best understood by referring to FIGURES 1-10 of the drawings, like numerals being used for like and corresponding parts of the various drawings.

Variable Descriptions, Values and Definitions

The following variable descriptions, values and definitions will be used to describe various features of the present invention.

Refill-data - Data stored in the Refill-data portion of a *getStructuredDexData* response. It could be $State_{Refill}$, delta (Δ) data between $State_{Refill}$ and $State_{Refill-Old}$ or other refill related information associated with the current state of a device.

Current-data - Data stored in the Current-data portion of a *getStructuredDexData* response. It could be $State_{Current}$, or delta (Δ) data between $State_{Current}$ and $State_{Refill-old}$ or other information related to the current state of a device.

$State_{Refill-database}$ - The refill state that is stored in the Network Operations Center (NOC) database. For a new device entry in the database, this value is preferably null (0). In the case where the NOC database has the latest refill state, $State_{Refill-database} = State_{Refill}$. In the case where the NOC database does not have the latest refill state, $State_{Refill-database} = State_{Refill-old}$.

$State_{Refill}$ - The most current refill state stored on the remote data acquisition and transmission device (RDATD). If the Controller on the RDATD has only been reset once, $State_{Refill} = State_{Refill-old}$.

$State_{Refill-old}$ - The refill state previous to the current refill state, i.e., $State_{Refill}$, stored on the RDATD. If the Controller has only been reset once $State_{Refill} = State_{Refill-old}$. $State_{Refill-Old}$ is also used as a reference state variable for a remote device.

$State_{Current}$ - The complete current state of a RDATD controller.

$DataLength_{Current}$ - Length of the Current-data block in a *getStructuredDexData* response:

If $DataLength_{Current} = 0$, there is no data for the current state.

If $DataLength_{Current}$ = FFFF, there is no change in current state since last retrieved.

If $DataLength_{Current}$ = xxx, the information contained in the Current-data block of the *getStructuredDexData* response is the actual length of the Current-data block.

$DataLength_{Refill}$ - Length of the Refill-data block in a *getStructuredDexData* response.

If $DataLength_{Refill} = 0$, there is no data for the current state.

If $DataLength_{Refill}$ = FFFF, there is no change in Refill-data since last retrieved.

If $DataLength_{Refill}$ = xxx, the information contained in the Refill-data portion of the *getStructuredDexData* response is the actual length of the Refill-data block.

$CRC_{Refill-database}$ - Cyclic Redundancy Check Value (CRC) for the Refill-data that was last received by the NOC and that is stored in the NOC database. For a new device, a

value of zero (0) is preferably stored in the database for this field.

CRC_{Refill} - the CRC for $State_{Refill}$, cached on the RDATD.

$CRC_{Refill-old}$ - the CRC for $State_{Refill-old}$, cached on the RDATD.

$\Delta_{Refill} = State_{Refill} - State_{Refill-old}$.

$\Delta_{Current} = State_{Current} - State_{Refill}$.

The term "wire-line transmissions" is used to refer to all types of electromagnetic communications over wires, cables, or other types of conduits. Examples of such conduits include, but are not limited to, metal wires and cables made of copper or aluminum, fiber-optic lines, and cables constructed of other metals or composite materials satisfactory for carrying electromagnetic signals. Wire-line transmissions may be conducted in accordance with teachings of the present invention over electrical power lines, electrical power distribution systems, building electrical wiring, conventional telephone lines, ethernet cabling (10baseT, 100baseT, etc.), coaxial cables, etc.

The term "wireless transmissions" is used to refer to all types of electromagnetic communications which do not require a wire, cable, or other types of conduits. Examples of wireless transmissions for use in local area networks (LAN) include, but are not limited to, radio frequencies, such as the 900 MHz and 2.4 GHz bands, infra-red, and laser. Examples of wireless transmissions for use in wide area networks (WAN) include, but are not limited to, radio frequencies, such as the 800MHz, 900MHz, and 1.9GHz ranges, infra-red, and laser.

FIGURE 1 is a block diagram of a system for communicating between a remote device and a network operations center incorporating teachings of the present invention. System 100 of FIGURE 1 preferably includes network operations center 126 communicatively coupled to wide area network (WAN) device 130 and local area network (LAN) device 134 via wide area network 124. Wide area network 124 can be either a wireless or a wire-line network.

System 100 can preferably utilize at least two different communication schemes for communicating between the network operations center 126 and WAN device 130 and/or LAN device 134. One communication scheme is the DEX/UCS protocol of data transfer as indicated at 138. The second communication scheme is a delta scheme for transmitting data from LAN device 134 and WAN device 130 to NOC 126 and vice versa as indicated at 142. The delta scheme of communication reduces the amount of data necessary to provide complete updated information to NOC 126 and database 230.

The delta scheme of the present invention utilizes a *getStructuredDexData* command to achieve this reduction in transmitted information. The *getStructuredDexData* command preferably selects records specified in a template from an original DEX/UCS data block associated with a remote device, restructures the records in a preferred order, and calculates a delta (Δ) or difference between a previous state and the current state of the remote device. Instead of sending the entire restructured data block, only the delta (Δ) is transmitted to NOC 126. In one embodiment, the delta is

compressed, using a conventional compression algorithm such as zip, gzip, etc., before transmitting the delta to the NOC 126. NOC 126 can recreate the current state of the remote device from delta (Δ) and values for a previous state that are stored in a database. The information associated with the various states of the remote device can include inventory levels, number of vends, condition of device hardware, as well as any other characteristic capable of being monitored and contained in the original DEX/UCS data block.

FIGURE 2 is a functional block diagram of one embodiment of a remote data acquisition system for vending machines, indicated generally at 210, according to the present invention. In general, system 210 of FIGURE 2 communicates information from a vending site 212 externally over a wide area wireless or wire-line network and internally over a local area wireless or wire-line network. As shown, the local area network at vending site 212 can be referred to as a device interrogation LAN subsystem (DIL). Vending site 212 may include only one vending machine 214 or a plurality of vending machines 214. Each vending machine 214 may include vending hardware (not expressly illustrated) and inventory 216 for performing vending functions and electronically tracking some vending information. Vending machines 214 may provide various types of products to customers such as soft drinks, snacks, etc.

According to the present invention, each vending machine 214 may include an application controller 218 coupled to and interfacing with vending hardware and inventory 216. Many vending machines 214 are equipped

with electronics for controlling vending operations as well as tracking some vending events such as money received, change given and number of vends from each slot. Application controllers 218 can communicate with such embedded electronics as well as be equipped to directly sense other vending events and vending equipment parameters (e.g. compressor performance). Application controllers 218 can also communicate with one another and the application host 222 via onboard transceivers using wire-line or wireless transmissions. According to the present invention, either the application controller 218 or the application host 222 can be configured to process the *getStructuredDexData* request or command, to restructure a DEX/UCS data block or to calculate delta (Δ) values.

Together, application controllers 218 and application host 222 form a LAN supported by the wire-line and/or wireless transmissions 220. In addition, application controllers 218 can also act as repeaters in case application host 222 cannot directly communicate with a particular application controller 218 while another application controller 218, which does have an established communication link with application host 222, can directly communicate.

Application host 222 acquires data captured by application controllers 218 and, preferably using the delta scheme of the present invention, can package and communicate that data across an external network 124 using a wide area network (WAN) interface. Application host 222 can be installed together with application controller 218 inside a vending machine or housed

separately in another location. In the event that the application host 222 is placed inside a vending machine together with an application controller 218, it is possible to share some of the electronic components between them, the LAN transceiver for example, in order to reduce the cost of the hardware. In this case, the application host 222 and application controller 218 inside the same vending machine, would preferably communicate with each other over a hardwired interface between the two components. Alternatively, the application host 222 and application controller 218 can be designed to be a single integrated component within a vending machine. Furthermore, an application host 222 can be used whose function preferably consists of monitoring the application controllers 218. For example, such an application host 222 could take the form of a hand-held portable computer 223 to be carried by service or delivery personnel in order to query the application controllers 218 without having to interact via the WAN interface 229. In one embodiment, application host 222 and/or application controller 218 may be used to perform the preferred functions associated with the automated or "Call-In" mode of operation mentioned above.

The WAN interface 229 can be implemented in a number of ways. In particular, WAN interface 229 is designed to support a wide area network 124 that can be implemented via wire-line or wireless transmissions. If a wireless narrowband PCS paging network is used to implement the WAN, messages from application host 222 can be communicated as digital messages through the paging

network, stored and delivered by the network carrier to the NOC using, for example, a secure Internet connection.

As shown in FIGURE 2, a network operations center (NOC) 126 communicates with one or more vending sites 212 across wide area network 124 using the delta scheme of the present invention. As mentioned, in one implementation, network operations center 126 can access information transmitted by application hosts 222 at vending sites 212 using the network carrier's infrastructure. In the embodiment of FIGURE 2, network operations center 126 includes a NOC control 228 that communicates with wide area network 124 through a WAN interface 229. NOC control 228 can receive data acquired from and transmit data to vending sites 212, process the data and store the data into database 230. NOC control 228 can also perform instant alert paging, direct dial alarms and other functions to provide real time notification to a vending operator upon the occurrence of certain events (e.g., out-of-stock, power outage, vandalism, etc.). NOC control 228 can also provide third party transaction processing such as allowing queries on database 230. The WAN interface 229 between NOC control 228 and the wide area network 124 can be implemented through the use of either wire-line or wireless transmissions.

At network operations center 126, a client access point 232 provides access from a client interface subsystem (CI) 234 across external network 224. In one implementation, client access point 232 can be a web-based interface allowing user access from a client computer across a network such as the Internet. Other

implementations include providing a direct-dial connection between client interface subsystem 234 and client access point 232. Once connected, a user can use client interface subsystem 234 to obtain information from database 230 based upon data acquired from vending sites 212. Further, users can be provided with extended services such as trend information developed by mining and analyzing database 230.

According to the present invention, system 210 of FIGURE 2 combines a number of technologies to provide technical advantages in the area of vending machine management, to reduce various operational costs and to overcome existing network traffic problems with conventional remote data acquisition systems for vending machines. As mentioned above, some conventional remote data acquisition systems employ a point-to-point wireless communication link to retrieve information from and send information to a plurality of remote devices. Further, wide-area networks (WAN) are often formed from a plurality of local area networks (LANs), and such LANs are often interconnected using a wire-line or wireless data transmission system. In other technical areas, wire-line and wireless transceivers have been used for local area network communication.

Delta scheme 142 of the present invention enables network data volume and communication time between NOC 126 and remote devices 130 and 134 to be minimized. Delta scheme 142 functions to minimize the amount of information necessary to be communicated between NOC 126 and devices 130 and 134 such that the complete state information of each device is maintained at NOC 126.

FIGURES 3A-3B illustrate one embodiment of the fields of a DEX/UCS block which has been restructured in response to a *getStructuredDexData* request. As illustrated in FIGURES 3A-3B, the DEX/UCS data block is preferably sectioned off into four categories. Category 305 preferably includes special fields, category 310 preferably includes fields that do not change frequently while category 315 preferably contains the fields that are likely to change frequently. Category 320 preferably includes the non-standard fields of a DEX/UCS data block. Restructuring the DEX/UCS data block allows for very high compression ratios to be achieved after the delta is calculated. These compression ratios may not be achievable without the restructuring of the DEX/UCS data block.

Software (not expressly shown) incorporating teachings of the present invention running on a device end, such as software running on application controller 218 or application host 222, will restructure the DEX/UCS data block according to a template framework, such as that illustrated in FIGURES 3A-3B, and by following a preferred set of rules. The preferred set of rules includes: to calculate Δ_{10}, $state_0$ is subtracted from $state_1$; if the DEX/UCS data block obtained from the RDATD controller does not contain a particular record type expected in the template, a character, such as a carriage return character (<CR>), is written to the restructured data block; if the data block from the RDATD controller contains a particular record type that is not expected in the template, it is ignored; for each record, only the fields of interest are considered (For example, for the

record "PA2*9888*543660*9882*543510" we may only need to send information "9888" and "543660," making our desired record "PA2*9888*543660."); for records that match, a <CR> is written to the restructured block; for records that don't match, the record identifier is skipped and a delta is calculated only for the remaining portion, (For example, for the two records "MA5*SEL1*1,7*9821,10086" and "MA5*SEL1*1,7*5696*5845," the delta is calculated for "1,7*9821*10086" and "1,7*5696*5845" portions only.); the delta is calculated on a per field basis, i.e., the fields separated by "*'s"; if a required field is absent in the DEX data block received from the RDATD controller, the restructured data block will have two contiguous "*'s" for that field; if all the bytes in the delta for a field are binary 0's (zeroes), the delta is considered to be empty and there is no delta data for that field to be written, (In this situation, there will be only two "*'s" in the record with no field value in between.); each such delta, except for the last record in line, is written to the restructured block followed by a "*"; the last record written to the restructured data block is followed by a <CR>; for fields that are not of equal length, e.g., "5845" and "10086," the shorter field is padded at the end with the appropriate number of 0's (zeroes) to make it equal in length to the longer field, (A delta is preferably calculated on two equal length fields.); since blank characters are allowed in the DEX/UCS data block, binary zeroes (0's) will be used for padding a shorter field to make it equal in length to the longer field, (This helps in reconstructing $state_1$ from $state_0$ and delta.); instead of "1 * 55", it is desirable to

minimize the size of the restructured data block and use "1*55" instead; by using 0 (zero) when adding the $state_0$ byte and the delta byte equals 0 (zero) we discard that byte since it was used for padding; and non-standard records are written to the very end of the restructured data block without calculating a delta.

FIGURES 4-8 illustrate one example of preferred steps processed by NOC 126 and device 400, such as a remote vending unit 214, during various *getStructuredDexData* requests. In FIGURES 4-8, the DEX data block is restructured at the remote device upon receipt of the *getStructuredDexData* request. Restructuring the DEX/UCS data block can also occur at other times during the processing of the *getStructuredDexData* request. In addition to calculating a delta in response to receipt of a *getStructuredDexData* request, a remote device may be configured to operate in an automated mode. This automated or "Call-In" mode is preferably configured such that a delta is calculated, generally as defined below, in response to a predetermined event, such as at a certain time, a threshold number of transactions, etc., and then transmitted to NOC 126.

FIGURE 4 illustrates the processing and transmissions which occur when NOC 126 transmits a *getStructuredDexData* request for $State_{Current}$ or the complete current state of device 400. As illustrated in FIGURE 4, NOC 126 transmits a *getStructuredDexData* request to get an update of the $State_{Current}$ of device 400. Included in the *getStructuredDexData* request for a $State_{Current}$ update, is the check value $CRC_{Refill-Database}$ as

indicated at 405. In response to receipt of the *getStructuredDexData* request for a $State_{Current}$ update, device 400 preferably writes $CRC_{Current}$ and $State_{Current}$ to a device response and then transmits the device response to NOC 126 as indicated at 410. In one embodiment, the information written to the device response is compressed prior to being written. Upon receipt of the device response containing $CRC_{Current}$ and $State_{Current}$, NOC 126 preferably recreates a current state from values stored in database 230 and the values of $CRC_{Current}$ and $State_{Current}$ provided in the device response.

FIGURES 5A-5C illustrate the processing which can occur in response to a *getStructuredDexData* request for the $\Delta_{Current}$ of device 400. FIGURE 5A illustrates one embodiment of the preferred steps that occur when updating database 230 with the changes which have occurred at device 400 since database 230 was last updated. As indicated at 505, to update database 230 with the current changes that have occurred at remote device 400, NOC 126 sends a *getStructuredDexData* request for $\Delta_{Current}$ to device 400. Included in the *getStructuredDexData* request for $\Delta_{Current}$ is error checking value $CRC_{Refill\text{-}Database}$. Upon receipt of the $\Delta_{Current}$ request and the $CRC_{Refill\text{-}Database}$ value, device 400 performs the steps indicated at 510. Device 400 begins by comparing the value of $CRC_{Refill\text{-}Database}$ provided by NOC 126 to a value of CRC_{Refill} accessible by device 400. A comparison of the values of $CRC_{Refill\text{-}Database}$ and CRC_{Refill} is performed to verify that NOC 126 and database 230 have the most current value for $State_{Refill}$ of device 400. If the values

of $CRC_{Refill-Database}$ and CRC_{Refill} are found to be equivalent, device 400 can then calculate $\Delta_{Current}$ by subtracting $State_{Refill}$ from $State_{Current}$ using a previously restructured data block or by restructuring a data block before calculating $\Delta_{Current}$. Device 400 will also preferably calculate a $CRC_{Current}$ value by applying a CRC function to $State_{Current}$. Once device 400 has completed all of the processing steps necessary to provide NOC 126 with the information requested, $CRC_{Current}$ and $\Delta_{Current}$ are written to a device response and transmitted to NOC 126 for processing as indicated at 515. The current state of device 400, the CRC calculated as well as other variables are stored by device 400 as previous state information for use with the next *getStructuredDexData* request once the device response has been transmitted.

Upon receipt of $CRC_{Current}$ and $\Delta_{Current}$ by NOC 126, database 230 is updated to reflect the current state of device 400. As indicated at 520, to update database 230, $\Delta_{Current}$ is added to the value of $State_{Refill-Database}$ stored in database 230 to recreate $State_{Current}$ or the current state of device 400. Once $State_{Current}$ has been stored, database 230 will then contain the current state of device 400. This updated information can be used to issue service calls, page a distributor to replenish inventory, or perform a myriad of other functions.

FIGURE 5B illustrates the processing which preferably occurs when $CRC_{Refill-Database}$ is compared to the value of CRC_{Refill}, during the processing of a *getStructuredDexData* request for $\Delta_{Current}$ by device 400, and the two are not equal. As indicated at 525, an

attempt by device 400 to interpret the value of $CRC_{Refill-Database}$ provided is made by comparing the value of $CRC_{Refill-Database}$ against the value of $CRC_{Refill-Old}$ that is available to device 400. If the value of $CRC_{Refill-Database}$ matches the value of $CRC_{Refill-Old}$, this indicates that the value of $CRC_{Refill-Database}$ provided by NOC 126 represents an older $State_{Refill}$ at NOC 126 than the latest $State_{Refill}$ transmitted by device 400. In such a situation, device 400 preferably provides $\Delta_{Current}$ and Δ_{Refill} to NOC 126 in order to update their corresponding values in database 230. As indicated at 525, Δ_{Refill} is calculated by subtracting $State_{Refill-Old}$ from $State_{Refill}$. $\Delta_{Current}$ is calculated as described above.

Once $\Delta_{Current}$ and Δ_{Refill} have been calculated, a device response is written, preferably using compressed data, and the update information is then transmitted to NOC 126. As indicated at 530, the information preferred to properly update database 230 includes $\Delta_{Current}$, Δ_{Refill}, CRC_{Refill}, $CRC_{Refill-Old}$ and $CRC_{Current}$. Upon receipt of $\Delta_{Current}$, Δ_{Refill}, CRC_{Refill}, $CRC_{Refill-Old}$ and $CRC_{Current}$ by NOC 126, database 230 is updated. As indicated at 535, the current refill state or $State_{Refill}$ of device 400 is calculated by adding Δ_{Refill} to $State_{Refill-Database}$ at NOC 126. The $State_{Refill}$ value is then stored as an updated $State_{Refill-Database}$ value. The current state or $State_{Current}$ of device 400 is recreated by adding $\Delta_{Current}$ to $State_{Refill}$. The new $State_{Current}$ value is then stored in database 230. Each CRC check value is also preferably stored in database 230 to update the check values each represents.

If device 400 determines that the value of $CRC_{Refill-Database}$ does not equal the value of CRC_{Refill} or $CRC_{Refill-Old}$, device 400 preferably transmits the complete $State_{Refill}$ and $\Delta_{Current}$ based on the current state of device 400. As illustrated at 540 of FIGURE 5C, $\Delta_{Current}$ is calculated by subtracting $State_{Refill}$ from $State_{Current}$. Once $\Delta_{Current}$ has been calculated, device 400 transmits $\Delta_{Current}$, $State_{Refill}$, $CRC_{Current}$ and CRC_{Refill} in a device response to NOC 126, as indicated at 545. Upon receipt, NOC 126 recreates and updates the appropriate variables stored in database 230.

To obtain the refill state or $State_{Refill}$ from device 400, NOC 126 may transmit a *getStructuredDexData* indicating such a request. As illustrated at 605 of FIGURE 6, a request for a $State_{Refill}$ update includes the transmission of $CRC_{Refill-Database}$. Similar to the request for the $State_{Current}$ update of FIGURE 4, device 400 preferably does not compare the value of $CRC_{Refill-Database}$ to any local CRC values. As indicated at 610, device 400 transmits CRC_{Refill} and $State_{Refill}$ to NOC 126 in response to the request for a $State_{Refill}$ update. Upon receipt of the device response containing the $State_{Refill}$ update, NOC 126 recreates the current state of device 400 based upon values stored in database 230 and the values of CRC_{Refill} and $State_{Refill}$. Database 230 is then updated accordingly.

Illustrated in FIGURES 7A-7C is the processing and transmissions which occur when NOC 126 transmits a *getStructuredDexData* request for Δ_{Refill} to device 400. As indicated at 705, transmitting a *getStructuredDexData* request for Δ_{Refill} preferably includes transmitting $CRC_{Refill-Database}$ to device 400 from NOC 126. Upon receipt

of the *getStructuredDexData* request for Δ_{Refill}, device 400 uses the $CRC_{Refill-Database}$ value supplied to verify that NOC 126 has the most current refill state or $State_{Refill}$ for device 400. If the value of $CRC_{Refill-Database}$ matches the value of CRC_{Refill} when compared, as illustrated at 710, device 400 can then transmit the information requested by NOC 126 in a device response. If the $State_{Refill}$ of device 400 has not changed since the last time device 400 updated database 230, device 400 transmits a $DataLength_{Refill}$ value equal to "FFFF," as indicated at 715, to NOC 126 to indicate that no change has occurred.

If device 400 compares the value of $CRC_{Refill-Database}$ to the value of CRC_{Refill} and determines the values to not be equal, as indicated at 720 of FIGURE 7B, device 400 will then compare the value of $CRC_{Refill-Database}$ to the value of $CRC_{Refill-Old}$. If the value of $CRC_{Refill-Old}$ matches the value of $CRC_{Refill-Database}$, indicating that the $State_{Refill}$ of device 400 has indeed changed since database 230 was last updated, Δ_{Refill} is calculated by subtracting $State_{Refill-Old}$ from $State_{Refill}$. Δ_{Refill} is then written to a device response and transmitted to NOC 126. In addition to Δ_{Refill}, CRC_{Refill} and $CRC_{Refill-Old}$ are also transmitted to NOC 126 in the device response as indicated at 725.

Should device 400 determine that the value of $CRC_{Refill-Database}$ transmitted by NOC 126 does not equal the value of CRC_{Refill} or the value of $CRC_{Refill-Old}$, as indicated at 730 of FIGURE 7C, device 400 will then transmit $State_{Refill}$ to NOC 126. In addition to $State_{Refill}$, device 400 transmits CRC_{Refill} and $CRC_{Refill-Old}$ to NOC 126 as

indicated at 735 such that database 230 can be updated accordingly.

FIGURE 8 illustrates one method of adding a new device to database 230. As illustrated at 805 of FIGURE 8, device 400 transmits unsolicited state information to NOC 126, i.e. in an automated or "Call-In" operating environment. Information included in an unsolicited transmission from a newly added device 400 might include CRC_{Refill}, $CRC_{Current}$, and $\Delta_{Current}$. The $\Delta_{Current}$ transmitted by device 400 is calculated by subtracting $State_{Refill}$ from $State_{Current}$.

Upon receipt of the unsolicited transmission indicated at 805, NOC 126 begins processing by comparing the value of CRC_{Refill} provided by newly added device 400 with the value of $CRC_{Refill-Database}$ in database 230 for device 400. Since, in this scenario, device 400 is new to the system, the value of $CRC_{Refill-Database}$ will be empty or zero (0). After determining that device 400 has recently been added to the system, NOC 126 transmits a *getStructuredDexData* request to device 400 as indicated at 810. In the *getStructuredDexData* request sent at 810, NOC 126 requests both $State_{Refill}$ and $\Delta_{Current}$ from device 400.

Device 400 responds to the receipt of the *getStructuredDexData* request from NOC 126 by transmitting the information requested. As indicated at 815, information included in a *getStructuredDexData* request for $State_{Refill}$ and $\Delta_{Current}$ preferably includes CRC_{Refill}, $CRC_{Current}$, $State_{Refill}$ and $\Delta_{Current}$.

Once NOC 126 receives the information requested, database 230 can then be updated as indicated at 820. Database 230 updates the value of $CRC_{Refill-Database}$ by setting its value equal to the value of CRC_{Refill} received. $State_{Refill}$ is also stored in database 230. The value of $State_{Current}$ in database 230 is created by summing $\Delta_{Current}$ and $State_{Refill}$.

An alternative to the method of FIGURE 8 for adding a new device to the system involves scheduling NOC 126 to transmit a *getStructuredDexData* request for $State_{Refill}$ and $\Delta_{Current}$ immediately after a new device is brought online. This proactive approach would eliminate the transmission which occurs at 805 of FIGURE 8 leaving only the processes and transmissions indicated at 810, 815 and 820.

FIGURES 9A-9B illustrates a flow chart indicating the preferred processing performed by device 400 upon receipt from NOC 126 or upon the automated execution of a *getStructuredDexData* request. Each of the scenarios encountered by device 400 in FIGURES 4-8 are generally processed according to method 900 of FIGURES 9A-9B.

Persons having ordinary skills in the art can appreciate the changes to FIGURES 4-9 which occur in a "Call-In" mode of generation. Upon receipt of the *getStructuredDexData* request from NOC 126, any information, such as return Node ID, $CRC_{Refill-Database}$, and flag information, included in the *getStructuredDexData* request is extracted, as indicated at step 905. Once the information has been extracted, the flag information is evaluated to determine if the *getStructuredDexData* request includes a request for the Refill-data

information of device 400. If it is determined, at step 910, that the *getStructuredDexData* request includes a request for the Refill-data of device 400, method 900 proceeds to step 915 to determine if the Refill-data request is a request for the $State_{Refill}$ or a request for the Δ_{Refill} of device 400. Alternatively, if at step 910 it is determined that the *getStructuredDexData* request received from NOC 126 does not include a request for the Refill-data of device 400, method 900 proceeds to step 917 where a $DataLength_{Refill}$ value equal to zero (0) is written to the device response. In a preferred embodiment of the present invention, data is compressed before being written to a device response.

At step 915, if it is determined that the *getStructuredDexData* request includes a request for Δ_{Refill}, method 900 proceeds to step 920 for a comparison of the CRC_{Refill} value of device 400 with the value of $CRC_{Refill-Database}$ provided by NOC 126. If the value of CRC_{Refill} is equal to the value of $CRC_{Refill-Database}$, method 900 proceeds to step 925 where a $DataLength_{Refill}$ value equal to "FFFF" is written in the device response. A $DataLength_{Refill}$ value equal to "FFFF" indicates to NOC 126 that there has been no change in the Refill-data since the last update requested from and transmitted by device 400. Once the device response has been written, method 900 proceeds to step 930.

Alternatively, if at step 920 the value of CRC_{Refill} is determined to be different than the value of $CRC_{Refill-Database}$, method 900 proceeds to step 935. At step 935, the value of $CRC_{Refill-Database}$ is compared to the value of $CRC_{Refill-Old}$. If the value of $CRC_{Refill-Old}$ equals the value

of $CRC_{Refill-Database}$, method 900 proceeds to step 940. At step 940, Δ_{Refill} is calculated by subtracting $State_{Refill-Old}$ from $State_{Refill}$. Δ_{Refill} is then written into a device response. Additionally, CRC_{Refill} is written in the device response to enable the value of $CRC_{Refill-Database}$ in database 230 to be updated. Upon completion of step 940, method 900 proceeds to step 930.

Should the value of $CRC_{Refill-Old}$ differ from the value of $CRC_{Refill-Database}$, method 900 proceeds from step 935 to step 945. If the value of $CRC_{Refill-Old}$ should differ from the value of $CRC_{Refill-Database}$, database 230 at NOC 126 will require a $State_{Refill}$ update. At step 945, a $State_{Refill}$ and a CRC_{Refill} value are written to a device response. Upon receipt of the device response at NOC 126, database 230 can then be updated with the values of CRC_{Refill} and $State_{Refill}$ provided. Upon completion of step 945, method 900 proceeds to step 930.

At step 930, the flags received in the *getStructuredDexData* request sent by NOC 126 are evaluated to determine if NOC 126 is requesting Current-data information from device 400. If, at step 930, it is determined that the *getStructuredDexData* request does not include a request for Current-data, method 900 proceeds to step 950 where a value of zero (0) is written in the device response for Current-data. Once step 950 has been completed, method 900 proceeds to step 955 where the response written by method 900 is transmitted to NOC 126.

Should it be determined at step 930 determine that the *getStructuredDexData* request includes a request for Current-data from device 400, method 900 proceeds to step 960. At step 960, it is determined whether the

getStructuredDexData request includes a request for a $\Delta_{Current}$ update or a request for a $State_{Current}$ update. If a $State_{Current}$ update is requested, method 900 proceeds to step 965 where $State_{Current}$ and $CRC_{Current}$ for device 400 are written a device response. Once $State_{Current}$ and $CRC_{Current}$ have been written to the device response at step 965, method 900 proceeds to step 955 where the device response is transmitted to NOC 126.

If a request for $\Delta_{Current}$ is included in the *getStructuredDexData* requested sent by NOC 126 as determined at step 960, method 900 proceeds to step 970. CRC_{Refill} is compared to the value of $CRC_{Refill-Database}$ at step 970. If the value of CRC_{Refill} is determined to equal the value of $CRC_{Refill-Database}$ at step 970, method 900 proceeds to step 975. At step 975, $\Delta_{Current}$ is calculated by subtracting $State_{Refill}$ from $State_{Current}$ and written to a device response as is a $CRC_{Current}$ value. Once $\Delta_{Current}$ and $CRC_{Current}$ have been written to the device response, method 900 proceeds to step 955 where the device response is transmitted to NOC 126.

Should it be determined at step 970 that the value of CRC_{Refill} does not equal the value of $CRC_{Refill-Database}$, method 900 proceeds to step 980 where the value of $CRC_{Refill-Old}$ is compared against the value of $CRC_{Refill-Database}$. If the value of $CRC_{Refill-Old}$ is determined to not equal the value of $CRC_{Refill-Database}$ at step 980, $State_{Refill}$ and CRC_{Refill} are written to a device response at step 985. If the value of $CRC_{Refill-Old}$ is determined to equal the value of $CRC_{Refill-Database}$ at step 980, Δ_{Refill} is calculated by subtracting $State_{Refill-Old}$ from $State_{Refill}$. Δ_{Refill} is then

written to the device response along with CRC_{Refill} at step 990. Upon completion of either step 985 or 990, method 900 proceeds to step 975 for the processing described above and then on to step 955 where the device response is transmitted to NOC 126. Based upon the above descrition, a person having ordinary skill in the art can appreciate the changes to FIGURE 4-9 which occur when device 400 is operated in a "Call-In" mode.

FIGURES 10A-10B illustrates a flow chart indicating the preferred processing performed by NOC 126 upon receipt of the device response created by device 400 in response to a *getStructuredDexData* request. Each of the scenarios encountered by NOC 126 in FIGURES 4-8 are preferably performed according to method 1000 of FIGURES 10A-10B. Upon receipt of the device response created by method 900, method 1000 preferably begins by extracting, such as uncompressing compressed data, the value of $DataLength_{Refill}$ as indicated at step 1005. Once the value of $DataLength_{Refill}$ has been obtained, method 1000 proceeds to step 1010 where $DataLength_{Refill}$ is compared against a null (0) character. If it is determined at step 1010 that the value of $DataLength_{Refill}$ is equal to the null (0) character, method 1000 proceeds to step 1015 where the value of CRC_{Refill}, provided in the device response created by method 900, is stored in database 230 as the value of $CRC_{Refill\text{-}Database}$. As a result, method 1000 is complete and the appropriate values of database 230 have been updated as indicated at 1020.

At step 1010, if it is determined that the value of $DataLength_{Refill}$ is something other than the null (0) character, method 1000 proceeds to step 1025. At step

1025, the value of $DataLength_{Refill}$ is compared to the value "FFFF". If the Refill-data of device 400 has not changed since the last device response transmitted by device 400, the value of $DataLength_{Refill}$ is equal to "FFFF" and method 1000 will then proceed to step 1020.

If, at step 1025, it is determined that the value of $DataLength_{Refill}$ does not equal "FFFF", method 1000 proceeds to step 1035. At step 1035, the values of $StateR_{efill}$, $Date/Time_{Refill}$, $Flag_{Refill}$, CRC_{Refill}, $CRC_{Refill-Old}$ and Refill-data are obtained. Once the desired values have been obtained, $Flag_{Refill}$ is tested at step 1040 to determine whether the Refill-data included in the device response is a $State_{Refill}$ update or Δ_{Refill} information. If $Flag_{Refill}$ indicates the information included in the device response is for a $State_{Refill}$ update, method 1000 proceeds to step 1045 where the Refill-data information and the value of CRC_{Refill} are stored in database 230. Once the storage is complete, method 1000 proceeds to step 1020 to repeat the method of FIGURES 10A-10B using Current-data vales and variables in place of Refill-data values and variables.

Alternatively, if it is determined at step 1035 that the value of $Flag_{Refill}$ indicates that Δ_{Refill} information is included in the device response received by NOC 126, method 1000 proceeds to step 1050. At step 1050, the value of $CRC_{Refill-Old}$ is compared to the value of $CRC_{Refill-Database}$. If the value of $CRC_{Refill-Old}$ does not equal the value of $CRC_{Refill-Database}$, method 1000 proceeds to step 1055 where a *getStructuredDexData* request for a $State_{Refill}$ update and $\Delta_{Current}$ is preferably generated and

subsequently transmitted to device 400 before NOC 126 ends current processing at 1060.

If it is determined that the value of $CRC_{Refill-Old}$ equals the value of $CRC_{Refill-Database}$ at step 1050, method 1000 proceeds to step 1065 where $State_{Refill}$ is calculated by summing Refill-Data and $State_{Refill-Database}$. Also at step 1065, $CRC_{Refill-Calc}$ is calculated by applying an appropriate CRC function to the value of $State_{Refill}$. Once a value of $CRC_{Refill-Calc}$ has been calculated, it is compared to the value of CRC_{Refill} at step 1070. The value of $CRC_{Refill-Calc}$ is compared to the value of CRC_{Refill} to determine if the information included in the device response received can be used to update the information maintained by database 230. If the value of $CRC_{Refill-Calc}$ does not equal the value of CRC_{Refill}, method 1000 proceeds to step 1055 for the processing described above and ends at 1060. If the value of $CRC_{Refill-Calc}$ equals the value of CRC_{Refill}, method 1000 proceeds first to step 1045 database 230 is updated and then on to 1020. Based on the above description, a person having ordinary skills in the art can appreciate the changes to FIGURES 4-10 when device 400 is operating in a "Call-In" mode.

Although the present invention has been described with respect to a specific preferred embodiment thereof, various changes and modifications may be suggested to one skilled in the art and it is intended that the present invention encompass such changes and modifications fall within the scope of the appended claims.